



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



Programming Distributed Computing Platforms with COMPSs

Pol Alvarez, Javier Alvarez, Ramon Amela, Rosa M. Badia, Javier Conejero, Marc Dominguez, Jorge Ejarque, Daniele Lezzi, Francesc Lordan, Cristian Ramon-Cortes, Sergio Rodriguez

Workflows & Distributed Computing Group

29-30/01/2019

Barcelona

Outline

Day 1

- Roundtable (9:30 – 10:00): Presentation and background of participants
- Session 1 (10:00 – 10:30): Introduction to COMPSs
 - Motivation
 - Setup of tutorial environment
- Session 3 (10:30-13:00): PyCOMPSs
 - Writing Python applications
 - Coffee break (11:00 – 11:30)
 - Python Hands-on using Jupyter notebooks
- Lunch break (13:00-14:30)
- Session 4 (14:30 -15:15): How to debug COMPSs applications
- Session 5 (15:15 -16:30): Python practical session (Bring your Own Code)
- SLIDES
 - http://compss.bsc.es/releases/tutorials/tutorial-PATC_2019/

Outline

Day 2

- Session 6 (9:30-11:00): COMPSs & Java
 - Writing Java applications
 - Java Hands-on
- Coffee break (11:00 – 11:30)
- Session 7 (11:30-12:30): COMPSs Advanced Features
 - Using binaries and MPI code
 - COMPSs execution environment
 - Integration with OmpSs
- Lunch break (13:30 – 14:30)
- Session 8 (14:30-15:30): Cluster Hands-on (MareNostrum)
- Session 9 (15:30 -16:30): Practical session (Bring your Own Code)
- COMPSs Installation & Final Notes



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

INTRODUCTION

Motivation

- New complex architectures constantly emerging
 - With their own way of programming them
 - Fine grain: e.g. APIs to run with GPUs, NVMs (Non-Volatile Memories)
 - Coarse grain: e.g. APIs to deploy in Clouds
 - **Difficult** for programmers
 - Higher learning curve / Time To Market (TTM)
 - What about non computer scientists???
 - **Difficult** to understand what is going on during execution
 - Was it fast? Could it be even faster? Am I paying more than I should? (**Efficiency**)
 - Tune your application for each architecture (or cluster)
 - E.g. partitioning data among nodes

Motivation

- Create tools that make user's life **easier**
 - Intermediate layer: let the difficult parts to those tools
 - Act on behalf of the user
 - Distributing the work through resources
 - Dealing with architecture specifics
 - Automatically improving performance
 - Tools for visualization
 - Monitoring
 - Performance analysis

BSC vision on programming models

Applications

Program logic independent of computing platform

PM: High-level, clean, abstract interface

General purpose
Task based
Single address space

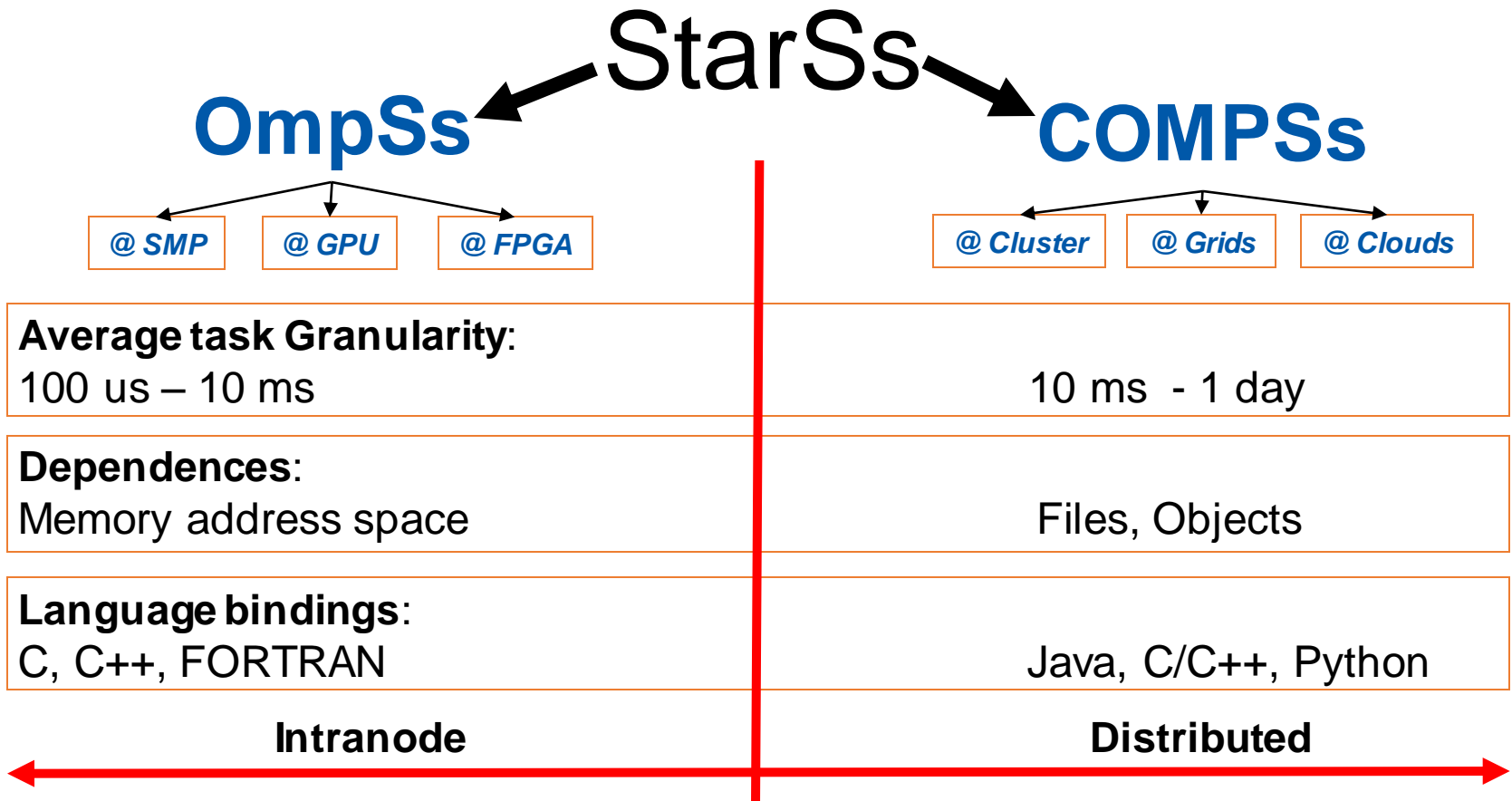
Power to the runtime

Intelligent runtime, parallelization, distribution, interoperability

API

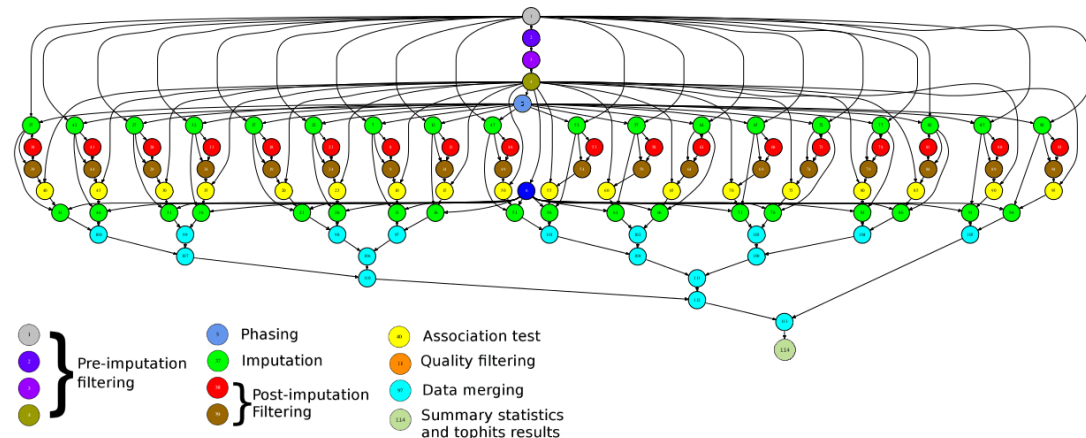


BSC vision on programming models



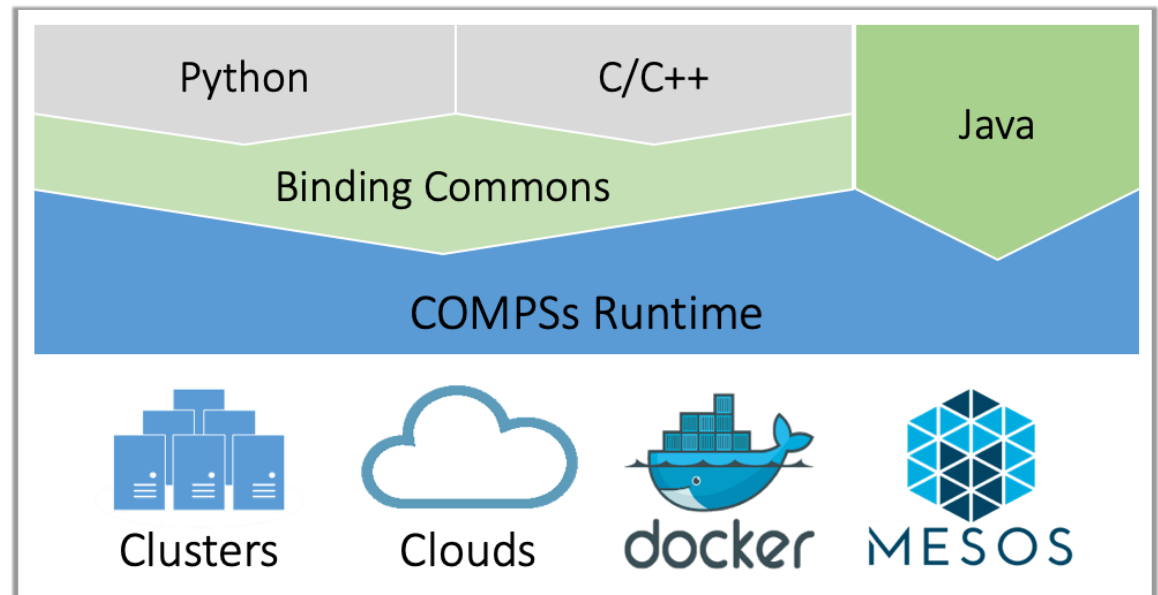
Programming with COMPSs

- Sequential programming
- General purpose programming language + annotations/hints
 - To identify tasks and directionality of data
- **Task based**: task is the unit of work
- Simple linear address space
- Builds a **task graph** at runtime that express potential concurrency
 - Implicit workflow
- Exploitation of parallelism
 - ... and of distant parallelism
- **Agnostic** of computing platform
 - Enabled by the runtime for clusters, clouds and grids

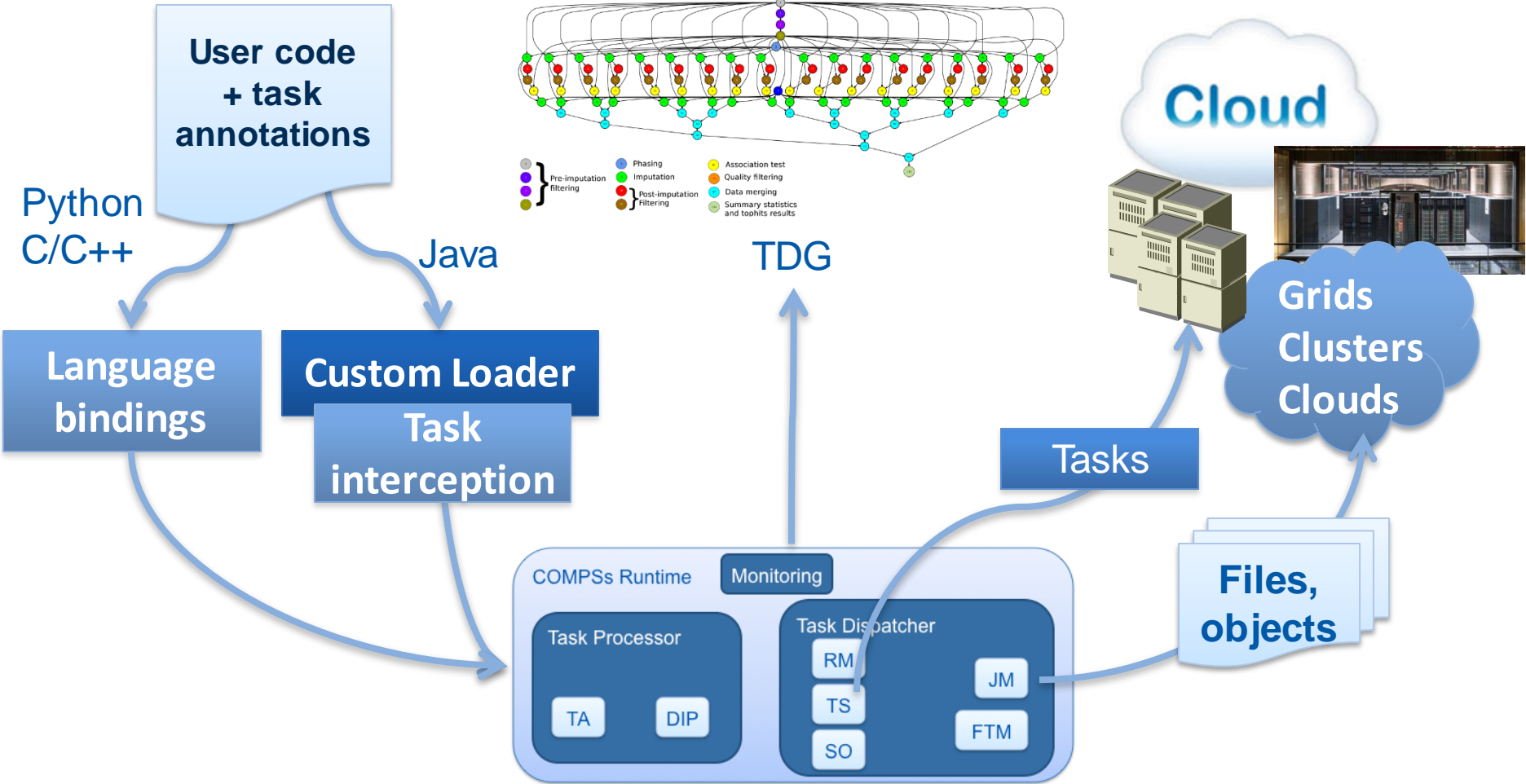


Programming with COMPSs

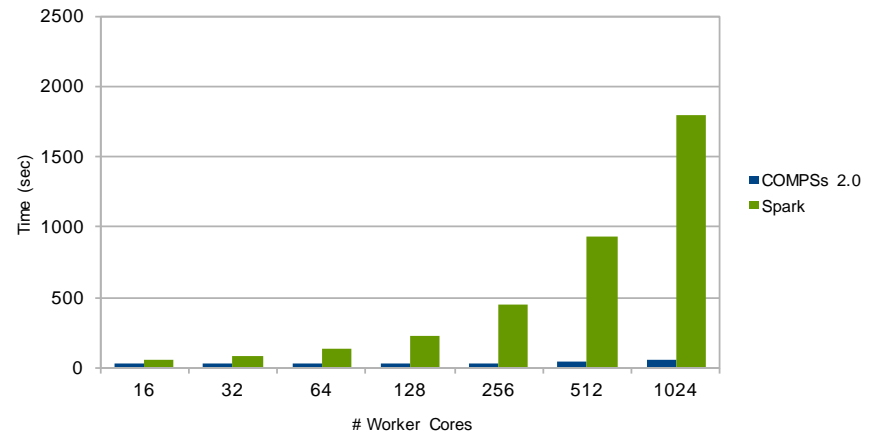
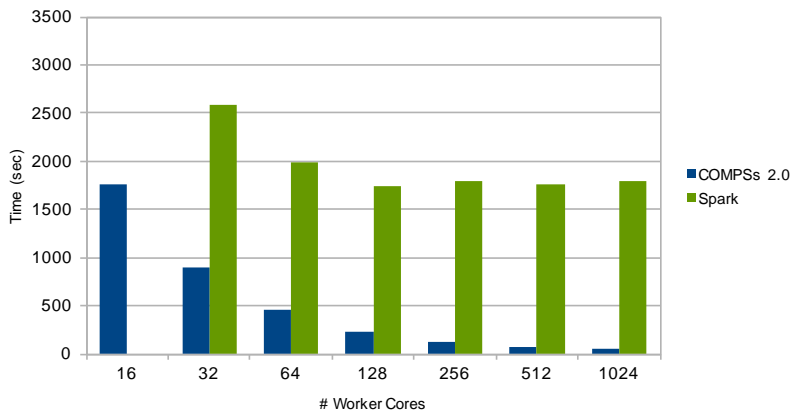
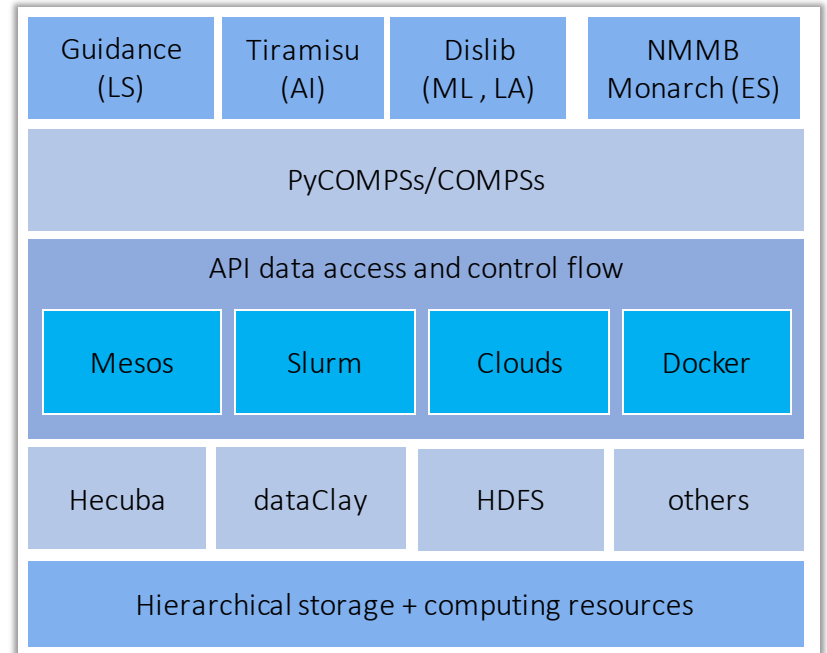
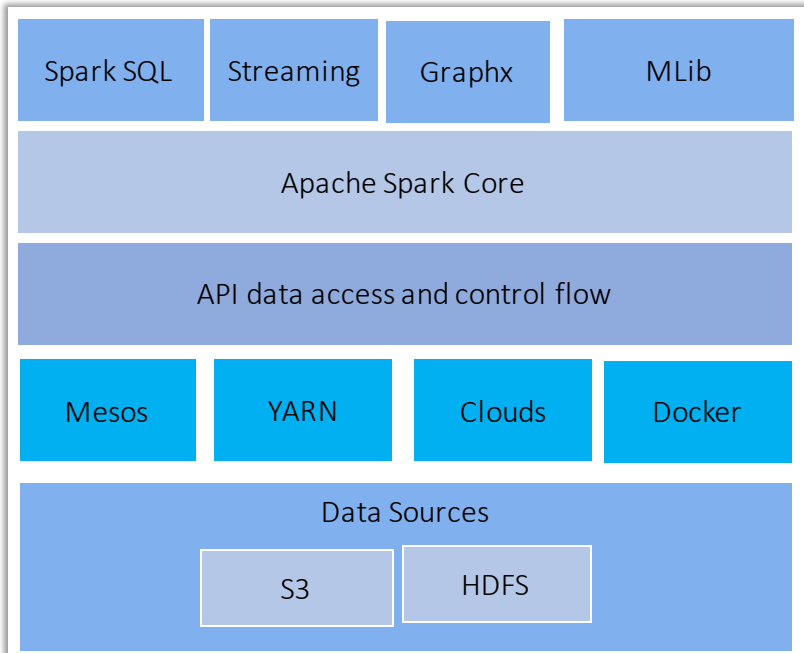
- Support for other types of parallelism
 - Threaded tasks (I.e., MKL kernels)
 - MPI applications -> tasks that involve several nodes
 - Integration with BSC **OmpSs**
- Available in MareNostrum, in the EGI Federated Cloud and in Chameleon Cloud



COMPSs Runtime

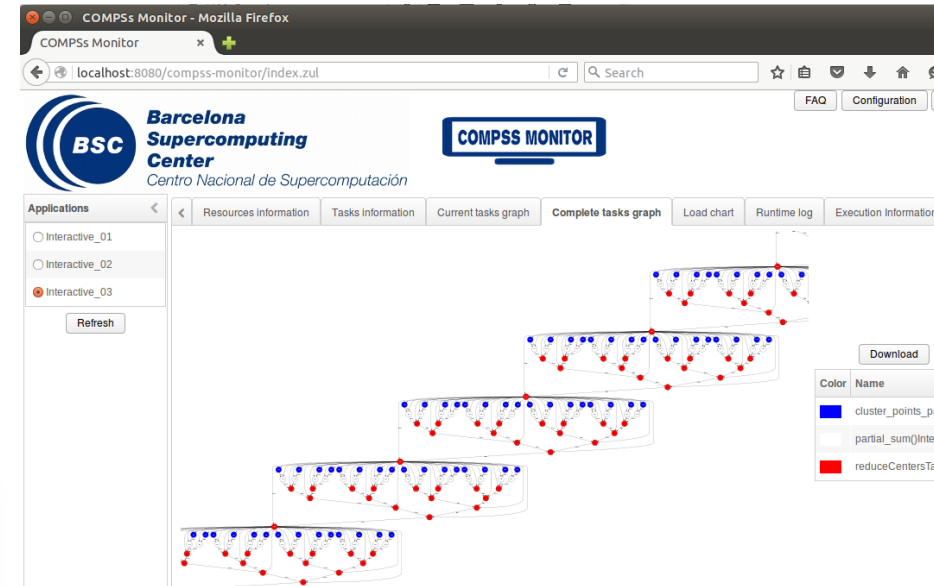
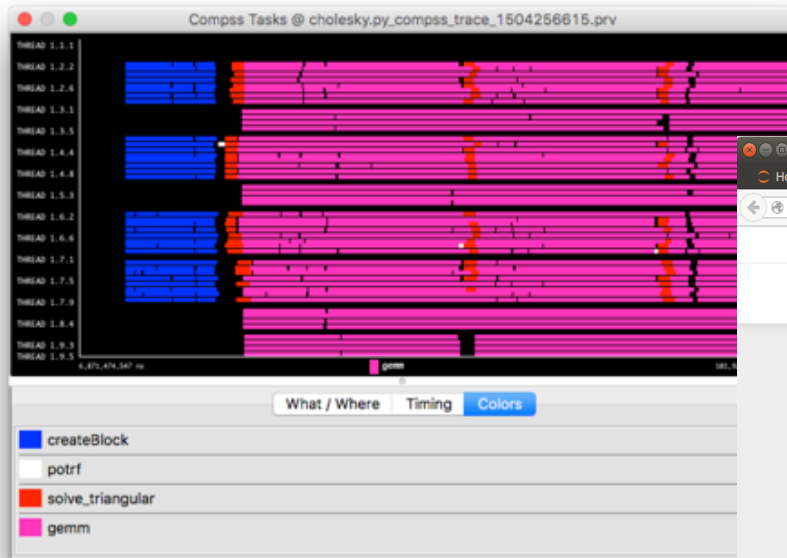


David vs Goliath



PyCOMPSs development environment

- Runtime monitor
- Paraver traces
- Jupyter-notebooks integration



kmeans-cool - Mozilla Firefox

Home

localhost:8888/notebooks/kmeans-cool.lipynb

jupyter kmeans-cool Last Checkpoint: a day ago (autosaved)

File Edit View Insert Cell Kernel Help Python 2

```
data.append(d)
return np.array(data)[:numV]
else:
return [np.random.random(dim) for _ in range(numV)]

In [7]: @task(returns=dict)
def cluster_points_partial(XP, mu, ind):
dic = {}
for x in enumerate(XP):
bestmukey = min([i[i[0], np.linalg.norm(x[1] - mu[i[0]])] for i in enumerate(mu)], key=Lam)
if bestmukey not in dic:
dic[bestmukey] = [x[0] + ind]
else:
dic[bestmukey].append(x[0] + ind)
return dic

Task appended.

In [8]: @task(returns=dict)
def partial_sum(XP, clusters, ind):
p = [(i, [(XP[j] - ind) for j in clusters[i]]) for i in clusters]
dic = {}
for i, l in p:
dic[i] = (len(l), np.sum(l, axis=0))
return dic

Task appended.
```

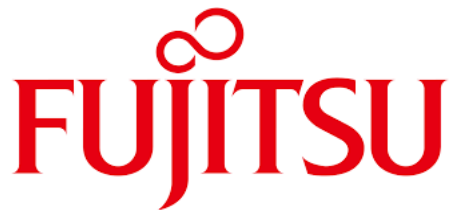
Projects where COMPSs is used/developed



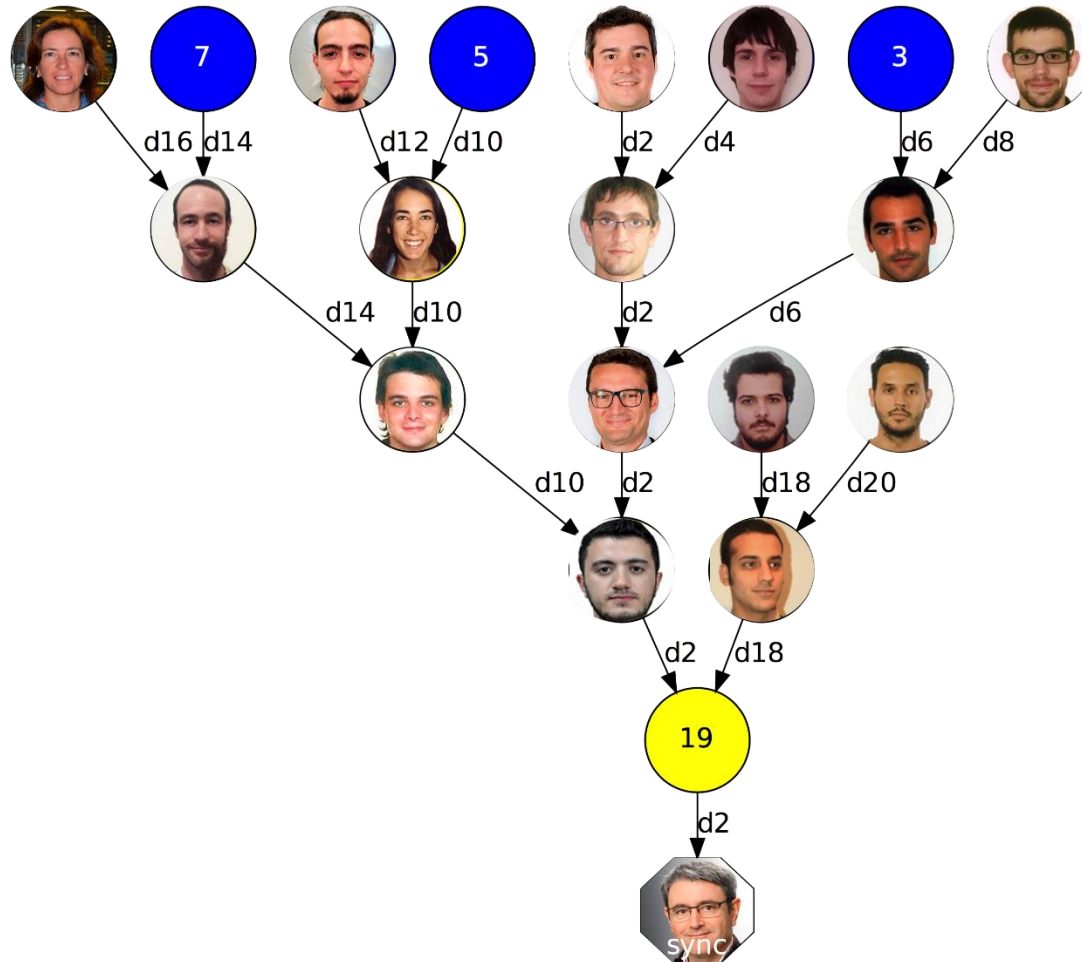
LANDSUPPORT



Exascale Quantification of Uncertainties for
Technology and Science Simulation



The WDC team





**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

SETUP OF THE TUTORIAL ENVIRONMENT

COMPSs docker

- Install Docker in your laptop
 - <https://www.docker.com/products/docker-desktop>

- Clone the examples apps

```
> git clone https://github.com/bsc-wdc/tutorial_apps.git
```

- COMPSs image downloadable from docker hub

```
> docker pull compss/compss-tutorial:patc2019
```

Laptop directory where you have the examples

- Start the container

```
> docker run -p 8888:8888 -p 8080:8080 -v path/to/tutorial_apps:/home/tutorial_apps \
  -itd compss/compss-tutorial:patc2019 --name mycompss
```

Image directory where you will find the examples

PyCOMPSs docker

- Start interactive session in the Docker container

```
> docker exec -it mycompss /bin/bash
```

Name of the container



PyCOMPSs docker

- Inside the image
 - Start the COMPSs monitor:

```
> /etc/init.d/comps-monitor start
```

- Start Jupyter

```
> jupyter-notebook --no-browser --allow-root --ip=172.17.0.2 --NotebookApp.token=
```

PyCOMPSs docker

- From your browser
 - Open Jupyter notebooks interface

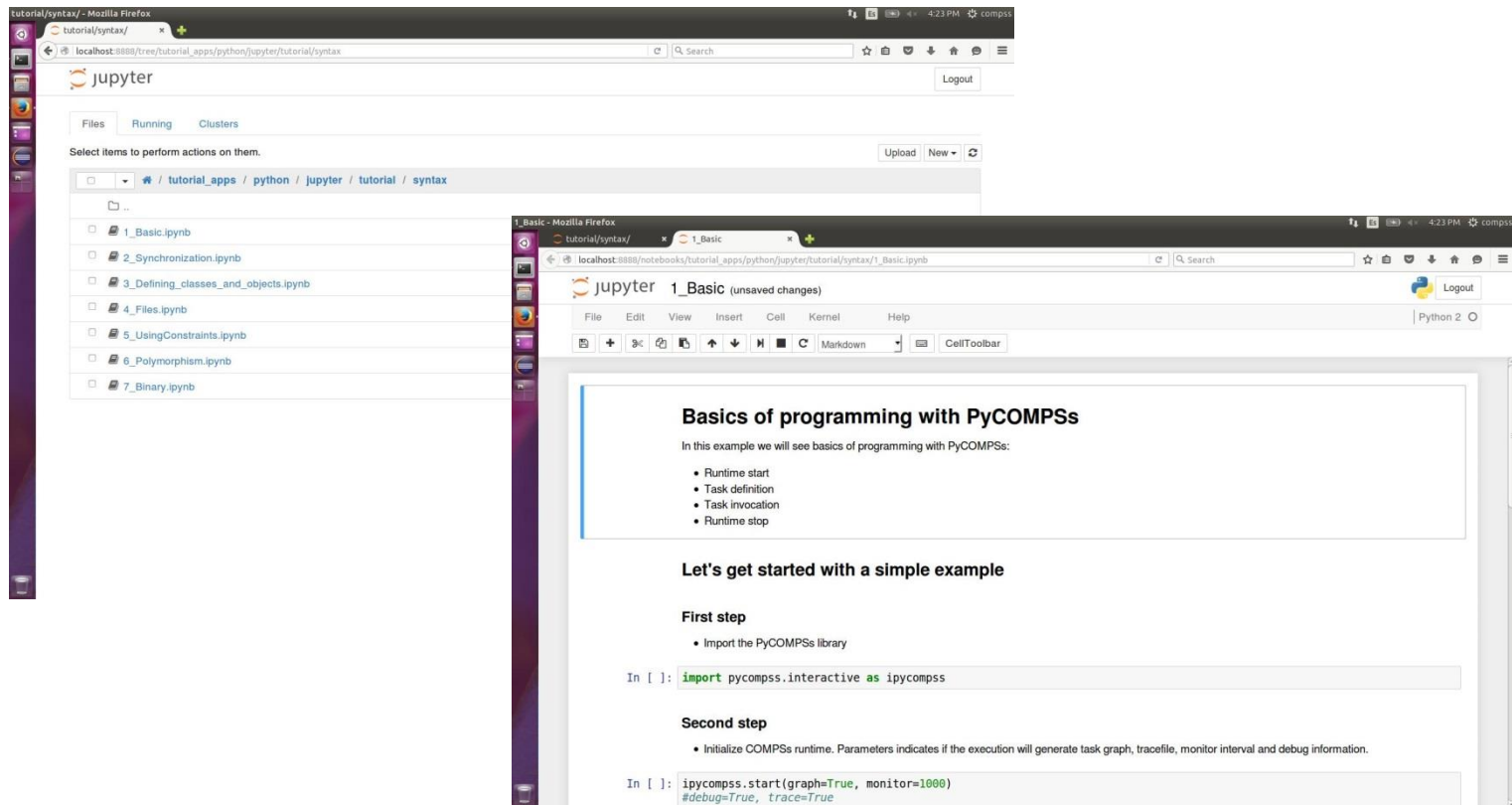
<http://localhost:8888/>

- Open COMPSs monitor

<http://localhost:8080/comps-monitor/index.zul>

PyCOMPSs docker

- Ready to play with the notebooks



The image shows a JupyterLab interface in a Mozilla Firefox browser. The left pane displays a file browser for the path `/tutorial_apps/python/jupyter/tutorial/syntax`, listing several IPython notebooks: `1_Basic.ipynb`, `2_Synchronization.ipynb`, `3_Defining_classes_and_objects.ipynb`, `4_Files.ipynb`, `5_UsingConstraints.ipynb`, `6_Polymorphism.ipynb`, and `7_Binary.ipynb`. The right pane shows the notebook `1_Basic` with the following content:

Basics of programming with PyCOMPSs

In this example we will see basics of programming with PyCOMPSs:

- Runtime start
- Task definition
- Task invocation
- Runtime stop

Let's get started with a simple example

First step

- Import the PyCOMPSs library

```
In [ ]: import pycomps.interactive as ipycomps
```

Second step

- Initialize COMPSs runtime. Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
In [ ]: ipycomps.start(graph=True, monitor=1000)
#debug=True, trace=True
```



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

PYTHON SYNTAX

Why Python?



- Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open. *
- Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C
- Large community using it, including scientific and numeric
- Object-oriented programming and structured programming are fully supported
- Large number of software modules available (>127,000 as of January 2018) **